1. Referent Object approach
2. Similar to Jones & Kelly but with a different data struct
   The invariant of Jones and Kelly approach is that every pointer is always in bounds.
3. We can find bounds by looking up the enclosing interval.

**Baggy Bounds Checking Representation:**

*Tricks:*
- Region of size \( l \) is aligned to \( 2^{\text{ceiling}(\log_2 l)} \)
- Allocation done in terms of power of 2.
- To share the bounds, we only need \( \log_2 \) of allocation size.

When we say "Aligned to \( 2^l \) bits", it actually means that the "\( l \) LSBs will be 0"

\[
\begin{align*}
\text{lo} &= p \& \sim (2^l - 1) \\
\text{hi} &= \text{lo} + 2^l
\end{align*}
\]

*Disadvantage:* This kind of memory allocation leads to memory fragmentation.
As can be seen above, it can waste up to half of the RAM size.

**Storing \( l \)**

Map \( p \rightarrow l \)
This is implemented as a special purpose hash table.

*Initialize:* byte \( L[2^{32}/\text{slotsize}] \)
\( l = L[p/\text{slotsize}] \)
Updates

```c
p=malloc(16);
L[p/slotsize]=4;
```

```c
p=malloc(32);
for(i=0; i<32/slotsize; i++)
    L[p/slotsize]=5;
```

In the Jones and Kelly approach, Updates take $O(\log N)$ time but they are independent of allocation size.

Here, the Updates depend on the allocation size but it is not a big drawback as the lookups occur far more often than the updates (and lookups are much faster in this approach).

Bound Checks

```c
q=p+1;
assert ((p XOR q)>>L[p/slotsize]) == 0);
if the above assertion is not 0,
q=OOB(p,q); (described later)
```

Check at dereference time:
```c
assert((q XOR (q+sizeof(*q)-1)>>L[q/slotsize])==0
*q=0;
```

Out Of Bounds Pointers (OOB)

When a pointer is out of bounds, we move it to the virtual memory (VM):
Set q=q|0x8000...0 (changing the MSB to 1)
**Given an out of bounds pointer, how can we access it?**

Using the Baggy Bound approach, we can access it only if it is in the immediate next half slot (below and above its own slot)

```c
if (q&0x8000...0)
    q'=q&0x7FFF...F)
```

If we assume that the pointer is within this range, then it can be safely said it lies in the grey shaded area only.

Therefore,
```c
if(q' % 16<8)
    l=L[q/slotsize-1];
else
    l=L[q/slotsize+1];
```

**But how do we know that the pointer is in these limits and not too far off?**

For this, we use OOB function to tell us that.

```c
void *oob(void *p, void *q)
{
    lookup_lo(p);
    lookup_hi(p);
    if((q>lo-slotsize/2 && q<lo) || (q>=hi && q< hi+ slotsize/2))
        return q|0x8000...0;
    abort;
}
```